Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования  
«Пермский национальный исследовательский политехнический университет»

(ПНИПУ)

**Факультет:** электротехнический

**Направление:** Промышленная мехатроника и робототехника

ОТЧЁТ

По лабораторной работе №11

«Классы»

Семестр 1

Выполнил:

студент группы ПРТ-21-1б

Поздняков Е.С.

Пермь 2022

**Задание №1**

**Постановка задачи**

Класс-контейнер МНОЖЕСТВО с элементами типа int,

Реализовать операции:

[] – доступа по индексу;

() – определение размера вектора;

- – разность множеств;

Пользовательский класс Money для работы с денежными суммами. Число должно быть представлено двумя полями: типа long для рублей и типа int для копеек. Дробная часть числа при выводе на экран должна быть отделена от целой части запятой.

Варианты реализации: при помощи стандартного типа данных и пользовательского класса

**Анализ**

1. Объявляем класс а с параметром размера множества. Пользователь вводит значения элементов множества.
2. Объявляем класс a1 из 5 элементов равных 7. Объявляем класс c и инициализируем его значением разности двух классов.
3. При помощи класса Iterator осуществляется вывод значений итератора i. При помощи перегруженных операторов инкремента и декремента осуществляется изменение значения i.

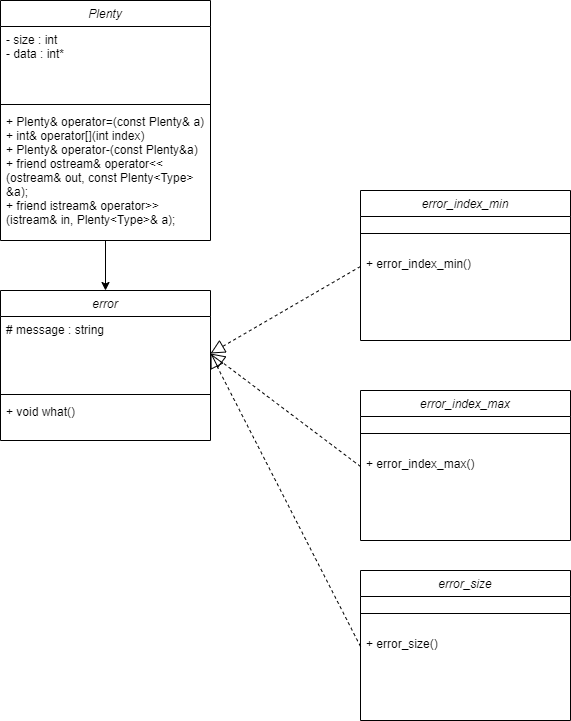
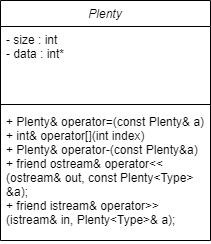
**Используемые переменные**

Plenty a – первое множество;

Plenty a1 – второе множество;

int с – значение элемента множества;

**UML-диаграмма**



**Код C++**

1 вариант:

#include <iostream>

using namespace std;

const int MAX\_SIZE = 30;

class Plenty

{

int size;

int\* data;

public:

Plenty(int s, int k = 0)

{

if (s > MAX\_SIZE) throw 1;

size = s;

data = new int[size];

for (int i = 0; i < size; i++)

data[i] = k;

}

Plenty(const Plenty& a)

{

size = a.size;

data = new int[size];

for(int i = 0; i < size; i++)

data[i] = a.data[i];

}

~Plenty()

{

delete [] data;

}

Plenty& operator=(const Plenty& a)

{

data = new int[size];

for(int i = 0; i < size; i++)

data[i] = a.data[i];

return \*this;

}

int& operator[](int i)

{

if (i < 0) throw 2;

if (i >= size) throw 3;

return data[i];

}

Plenty& operator-(const Plenty&a)

{

if (size != a.size)

{

return \*this;

}

else

{

for (int i = 0; i < size; i++)

data[i] = data[i] - a.data[i];

return \*this;

}

}

friend ostream& operator<<(ostream& out, const Plenty&a);

friend istream& operator>>(istream& in, Plenty& a);

};

ostream& operator<<(ostream& out, const Plenty&a)

{

for (int i = 0; i < a.size;i++)

out << a.data[i] << " ";

return out;

}

istream& operator>>(istream& in, Plenty& a)

{

cout << "Input size: ";

cin >> a.size;

for (int i = 0; i< a.size; i++)

{

cout << "Input elem: ";

in >> a.data[i];

}

return in;

}

int main()

{

try

{

Plenty a(2);

cout << "Plenty a" << endl;

cin >> a;

int index;

cout << "Input index: "; cin >> index;

cout << "a[" << index << "] = " << a[index];

}

catch (int)

{

cout << "Error!" << endl;

}

return 0;

}

2 вариант

#include <iostream>

using namespace std;

class error

{

protected:

string message;

public:

void what() {cout << message;}

};

class error\_index\_min: public error

{

public:

error\_index\_min()

{

message = "Index fewer than 0";

}

};

class error\_index\_max: public error

{

public:

error\_index\_max()

{

message = "Index greater than max index";

}

};

class error\_size : public error

{

public:

error\_size()

{

message = "Error, sizes don't match";

}

};

class Plenty

{

private:

int size;

int\* data;

public:

Plenty(int s, int k = 0)

{

size = s;

data = new int[size];

for (int i = 0; i < size; i++)

data[i] = k;

}

Plenty(const Plenty& a)

{

size = a.size;

data = new int[size];

for(int i = 0; i < size; i++)

data[i] = a.data[i];

}

~Plenty()

{

delete [] data;

}

Plenty& operator=(const Plenty& a)

{

data = new int[size];

for(int i = 0; i < size; i++)

data[i] = a.data[i];

return \*this;

}

int& operator[](int index)

{

if (index < 0) throw error\_index\_min();

if (index >= size) throw error\_index\_max();

return data[index];

}

Plenty& operator-(const Plenty&a)

{

if (size != a.size)

{

throw error\_size();

return \*this;

}

else

{

for (int i = 0; i < size; i++)

data[i] = data[i] - a.data[i];

return \*this;

}

}

friend ostream& operator<<(ostream& out, const Plenty&a);

friend istream& operator>>(istream& in, Plenty& a);

};

ostream& operator<<(ostream& out, const Plenty&a)

{

for (int i = 0; i < a.size;i++)

out << a.data[i] << " ";

return out;

}

istream& operator>>(istream& in, Plenty& a)

{

cout << "Input size: ";

cin >> a.size;

for (int i = 0; i< a.size; i++)

{

cout << "Input elem: ";

in >> a.data[i];

}

return in;

}

int main()

{

try

{

Plenty a(2);

cout << "Plenty a" << endl;

cin >> a;

Plenty a1(5, 7);

cout << "Plenty a1 : " << a1 << endl;

Plenty c = a1-a;

cout << "a1 - a = " << c << endl;

int index;

cout << "Input index: "; cin >> index;

cout << "a[" << index << "] = " << a[index];

}

catch (error& a)

{

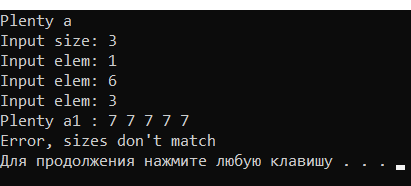
a.what();

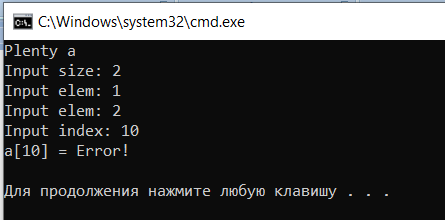
}

return 0;

}

**Скриншот результата**





**Анализ результатов**

Пользователь вводит значения элементов первого множества. Из элементов второго множества вычитаются элементы первого множества и выводятся на экран.

**Контрольные вопросы**

1. Что представляет собой исключение в C++?

Исключение – это объект, который система должна генерировать при возникновении исключительной ситуации. Генерация такого объекта создает исключительную ситуацию.

2. На какие части исключения позволяют разделить вычислительный процесс? Достоинства такого подхода?

Исключения позволяют разделить вычислительный процесс на 2 части:

- обнаружение аварийной ситуации

- обработка аварийной ситуации

Достоинства такого подхода:

- удобно использовать в программе, которая состоит из нескольких модулей

- не требуется возвращать значение в вызывающую функцию

3. Какой оператор используется для генерации исключительной ситуации?

При помощи throw генерируется исключение

4. Что представляет собой контролируемый блок? Для чего он нужен?

Контролируемый блок используется для проверки возникновения исключения

5. Что представляет собой секция-ловушка? Для чего она нужна?

Секция-ловушка нужна для обработки исключений

6. Какие формы может иметь спецификация исключения в секции ловушке? В каких ситуациях используются эти формы?

Catch может иметь 3 формы:

- (тип имя)

- (тип)

- (…)

Формы 1 и 2 обрабатывают конкретные исключения, а форма 3 перехватывает все исключения.

7. Какой стандартный класс можно использовать для создания собственной иерархии исключений?

Класс exception

8. Каким образом можно создать собственную иерархию исключений?

Можно определять собственные исключения, унаследовав их от класса exception

9. Если спецификация исключений имеет вид: void f1() throw(int, double); то какие исключения может прождать функция f1()?

Функция генерирует те исключения, которые явно указаны в этом списке

10. Если спецификация исключений имеет вид: void f1() throw(); то какие исключения может прождать функция f1()?

Функция исключений не генерирует

11. В какой части программы может генерироваться исключение?

В конце